Application

for

United States Letters Patent

To all whom it may concern:

Be it known that

ANDREW WASON
MICHAEL MILLS
CHRIS O'BRIEN
BRUCE A. WALLACE

have invented certain new and useful

CROSS-PLATFORM FRAMEWORK-INDEPENDENT SYNCHRONIZATION ABSTRACTION LAYER

of which the following is a full, clear, concise, and exact description.

EV243779439US

CROSS-PLATFORM FRAMEWORK-INDEPENDENT SYNCHRONIZATION ABSTRACTION LAYER

APPENDIX AND COPYRIGHT NOTICE

This document includes a partial source code listing of a preferred embodiment of the applicant's invention. The code, listed in Appendix A and in the drawings, forms part of the disclosure of the specification. It is copyrighted. The copyright owner has no objection to facsimile reproduction by anyone of the patent document, including the copyrighted Appendix A and the drawings as they appear in the Patent and Trademark Office file or records, but otherwise reserves all rights.

TECHNICAL FIELD

This invention relates to the field of software plug-ins for multimedia file players and for other applications supporting ordered data flow files. More precisely, this invention defines a new field of software that allows plug-ins and content to be insulated from differences in underlying platforms and frameworks.

BACKGROUND OF THE INVENTION

Multimedia player frameworks have become widespread. Later versions of the most popular Internet browsers - Microsoft's Internet Explorer and Netscape's Communicator - include at least one player in the basic package. These are frameworks such as

RealNetworks, Inc.'s RealPlayer™ G2 family (collectively, the "RealPlayer™"); Microsoft Corporation's Windows Media Technologies (NetShow™); Macromedia, Inc.'s Director™; Apple Computer, Inc.'s QuickTime™; and Sun Microsystems, Inc.'s Java™ Media Framework (JMF). Most of these frameworks are extensible by means of plug-ins discussed below; some, notably JMF, are extensible by means of applications built on top of the frameworks. By being extensible we mean that a particular framework supports a set of interfaces exposed for interaction with additional software modules or components.

A framework is a set of interfaces, *e.g.*, API's and procedures, and a set of capabilities exposed for the purposes of extensibility. It need not operate on multimedia-type files, *i.e.*, files ordered sequentially and supporting the concept of a time-line; frameworks generally need not operate on time- or otherwise-ordered data files. In the rest of this document, however, we will discuss predominantly frameworks operating on *ordered data flow* files. We will refer to such frameworks interchangeably as "frameworks," "extensible frameworks," "extensible data flow-based frameworks," or by similar expressions; we do not imply any difference in the particular designation used, unless otherwise indicated.

Frameworks are extended by means of extensions, for example plug-ins. Plug-ins, also referred to as extension modules, are

modules or components dynamically loaded at run-time. Extensible architecture and plug-ins have been used in commercial products, the RealPlayer™ and Windows Media Technologies being two instances. They also have been described in various publications. See United States Patents No. 5,838,906 and 5,870,559. (The specifications of these patents are incorporated by reference as if fully set forth herein.) A number of companies market plug-ins specifically for extending multimedia players.

OBJECTS OF THE INVENTION

One difficulty faced by plug-in developers is that a plug-in must be ported for each platform, *i.e.*, for each hardware-operating system combination. Another difficulty lies in adapting a plug-in to various frameworks. Third difficulty, closely related to the first two, is that platform porting and framework adaptation require developers to have working knowledge of the various platforms and frameworks. One object of this invention is to provide a single interface that allows plug-in developers to build a single, platform independent version of a plug-in. Another object of this invention is to provide a uniform interface to the various frameworks so that a single plug-in can extend functionality of multiple frameworks. The third object of this invention is to facilitate development of content that can operate similarly, *i.e.*, consistently, with different multimedia players

(e.g., RealNetworks, Inc.'s RealPlayer™ and Microsoft Corporation's Windows Media Technologies (NetShow™)), and platforms (TV, PC, set-top boxes). The fourth object of this invention invention is to accelerate the development of content and multimedia plug-ins by promoting reuse of existing software objects (*e.g.*, objects written in Java™ and JavaScript).

SUMMARY OF THE INVENTION

This invention is an abstraction layer providing a uniform interface between a framework and one or more plug-ins. In the preferred embodiment, the invention is a Synchronization Abstraction Layer (SAL) abstracting time-based frameworks into a common synchronization interface. The SAL synchronizes itself and other plug-ins to a time-line of the underlying framework - and it does that independently of the underlying framework. In other words, the plug-ins interact with the underlying framework through the SAL, rather than directly. Typically, the SAL is implemented on top of the synchronization of the Application Programming Interfaces (API's) provided by the underlying frameworks. It has at least one point of coupling with the underlying framework: a method for providing the SAL with the current time.

The SAL includes a uniform cross-platform interface for the plug-ins. Preferably, the cross-platform interface has a functional core independent of the specific framework underlying

the SAL.

Some of the features of this invention are listed below:

1. Secure e-commerce, full interactive experiences including text and voice chat, games, graphics, animations, and advertising can be integrated and synchronized with streaming multimedia such as video and audio;

2. It can be used to build a content framework that insulates content and plug-in developers from details and differences in hardware platforms, so that the same content or plug-in can run on desktop platforms (PC, Macintosh, Solaris, Linux), Televisions (GI, SA), or other kinds of devices (AOL-TV, screen phones);

3. It can be used to build a content framework that insulates content and plug-in developers from specifics of, or differences in, software platforms, so that the same content or plug-in can run on Microsoft NetShow™, RealNetworks RealPlayer™, Apple Quicktime™, Sun Java™ Media Framework or any other media system;

4. It can run without a network or on different types of networks, such as wireless, Intranets, or the Internet;

5. It can be used to synchronize arbitrary data types,

including text, chat, graphics, video, audio, and active content like Java™, JavaScript, or Flash;

6. The framework-independent layer can be implemented using different languages, including Java™, HTML, XML, JavaScript, VBScript, or other languages;

7. When used for video synchronization, it can be used to synchronize arbitrary datatypes with different spatial objects in video, and with different temporal objects in video.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a diagram of basic processes in a traditional extensible time-based framework of prior art.

Figure 2 is a diagram of basic processes in an extensible time-based framework incorporating the Synchronization Abstraction Layer in accordance with this invention.

Figure 3 is a high level diagram of a preferred embodiment implementation specific to a table of contents plug-in.

Figure 4 lists a sample toc.rj, an XML descriptor file for a table of contents plug-in of the previous Figure.

Figure 5 lists a sample toc.xml, an XML descriptor file with detailed hierarchical description of the table of contents plug-in of Figure 3.

Figure 6 lists a sample toc.smi, an XML file defining a set of synchronized plug-ins.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 depicts conventional operation of an extensible multimedia player framework connected to a media server. Server 110 includes Media Object 111, Clock Object 112, and Custom Objects 113. Framework 120 and Extension Module (plug-in) 130 coexist on the client side. As is usual for a media player, Framework 120 contains Video Media Object 121, Clock Object 122, and Audio Media Object 123. Extension Module 130 has Custom Objects 131 and 132. Server 110 connects to Framework 120 and Extension Module 130 through Network 150. Line 160 indicates the extensibility interface between Framework 120 and Extension Module 130. Note that this high level diagram of prior art omits many other components and processes, including an operating system and rendering means.

Figure 2 demonstrates an arrangement functionally similar to the arrangement of Figure 1, but using the present invention. Server 210, including Media Object 211, Clock Object 212, and Custom Objects 213, connects to Framework 220, Extension Module (plug-in) 230, and Synchronization Abstraction Layer (SAL) 270 through Network 250. Line 260 symbolizes the interface extending Framework 220 through SAL 270. Block 271 denotes Uniform

Interface between SAL 270 and various extension modules, including Extension Module 230.

As before, Figure 2 omits many other components and processes, and is rather generalized. But a number of structural changes can be envisioned even in this, high level representation. For instance, media files can be stored locally, obviating the need for transmission over the network. Further, custom objects need not reside on the same server as the media files; they may be located on a separate server or, in some cases, locally. Several extension modules can be simultaneously connected to the same client framework, either through the SAL or directly. (The SAL itself is, of course, an extension module.)

The discussion above focused on media players rendering multimedia files supporting the concept of time-line, such as video and audio clips, because these applications are commercially important. The invention, however, is not limited to rendering multimedia or time-line ordered files. Time-line sequencing is only one example of data flow order within files, including media files. Many other types of ordering are possible. As one example, consider a video taken from a window of a car going down a highway. The images can be time- or distance-sequenced.

Files of interest in the present context, which we designate as ordered data flow files, may also be ordered in multiple

dimensions. Imagine a video game called "Tank Commander," with the player controlling a virtual tank moving cross country. Because the tank can move in two dimensions, the top view of the tank and its immediate surroundings depends on two coordinates. The video data must therefore be ordered in two dimensions. Similarly, video of a one-directional view from an object moving in three dimensions requires three dimensional ordering. Clearly, the concept of ordered files goes beyond the four dimensions of space-time.

We also need not limit this concept to files ordered according to variables perceived as continuous, *e.g.*, space and time. The concept is equally applicable to ordering according to discrete variables, such as event occurrences or quantized space positions. As an example, consider a server storing a library of books and sending the content to the client for sequential viewing page by page. The concept of ordered data flow files is therefore a general one.

Finally, in its general form the invention need not synchronize data flow, but may merely provide a uniform cross-platform interface for plug-ins.

Although the invention can be implemented in various languages, it is preferable to use a platform-independent source code. Therefore, consistent with the spirit of the invention, its preferred embodiment is implemented in Java™ - an

architecture-neutral, object-oriented, multithreaded language intended for use in distributed environments. (Partial source code of the preferred embodiment is listed in Appendix A and constitutes a part of the disclosure of this specification.) The Java™ code is executed by a Java™ Virtual Machine (JVM), which is a byte-code interpreter. The JVM is stored in one location and instantiated separately for each extension module, eliminating the need to store duplicate copies of the JVM for other modules that are also implemented in Java™. Alternatively, the invention can instantiate a JVM from a copy in the framework, in the operating system, or even in another application.

The preferred embodiment creates a uniform interface to the RealPlayer™ of Real Networks. It can be easily adapted to extend other multimedia players in addition to the RealPlayer™, especially if a common set of functional features exists in the various players. This should be the case for most multimedia players. Because the Java™ code of a specific extension can be mostly independent of a particular framework being extended, large sections of the SAL's code can be shared among different framework adaptations of the same plug-in.

Figure 3 illustrates a high level diagram of a more specific example of the preferred embodiment. Here, the SAL is used to plug-in a table of contents ("TOC") extension module into the

RealPlayer™. The TOC plug-in displays a hierarchical view of the table of contents of a video presentation described in the table of contents. In the hierarchy, entries in the table are highlighted during the video presentation of their associated video portions. A user can randomly access various portions of the video by clicking on the corresponding entries in the TOC.

In this configuration RealPlayer™ 301 has three extension modules: SAL (RealJava) 310, RealVideo 302, and RealText 303. (Hereinafter we denominate by "rj" or "RealJava" all Java™ objects compatible with the G2 architecture.) For clarity, SAL 310 is expanded within the dotted oval into its conceptual components - File Format Filter 311 and Renderer 312. This conceptual separation results from the G2 architecture constraints of the RealPlayer™; in a SAL adaptation compatible with another media player, such conceptual separation may be inappropriate.

When a user invokes the RealPlayer™ application and the TOC extension module, RealPlayer™ 301 accesses Web Server 320 over Network 330 to retrieve "toc.smi," a ***s***ynchronized ***m***ultimedia ***i***ntegration ***l***anguage ("SMIL") descriptor file defining a set of synchronized plug-ins to run simultaneously in the RealPlayer™ environment. This file contains a reference to "toc.rj," an e***x***tensible ***m***arkup ***l***anguage ("xml") descriptor file for the table of contents extension module compatible with the RealPlayer™.

Based on the MIME type of the toc.rj file, RealPlayer™ 301 instantiates SAL 310 and routes the toc.rj file to it. SAL 310 includes File Format Filter 311 and Renderer 312, each implemented using Java™ code linked with native platform C++ code.

File Format Filter 311 parses the toc.rj file and hands it over to Renderer 312 through a standard protocol defined by the RealPlayer™. The parsed data contains following elements: (1) name of the Java™ class to instantiate; (2) location from which the Java™ class can be retrieved; and (3) any other parameters specific to the extension module. A sample toc.rj file appears in Figure 4. The first line (starting with *classid*) specifies the Java™ class names; the next two lines (*codebar* and *archive*) specify a URL location and names of Java™ class files; the following line assigns the *width* and the *height* of the window for the table of contents; *sync* is the period in milliseconds between synchronizing events, *i.e.*, between timing calls from the RealPlayer™ to the SAL; *duration* refers to total length of the time-line of the presented event; and, lastly, *param* is the parameter that points to the URL with the toc.xml file, the XML file descriptor of the specific table of contents.

Renderer 312 instantiates JVM (Java™ Virtual Machine) 313 needed to run the Java™ code, and also retrieves ".jar" files (packaged Java™ class files) over Network 330. Note that the

URL for the ".jar" files is available from the parsed toc.rj file discussed above. Renderer also instantiates an instance of RealTOC object 314 and hands to it the URL for toc.xml, the XML file describing the specific table of contents object associated with the multimedia presentation. This file includes the nodes and sub-nodes within hierarchical structure of the specific table of contents; it also includes the "begin" and "end" times corresponding to each node and sub-node. RealTOC retrieves the toc.xml file, parses it, and builds the specific table of contents tree structure. A sample toc.xml file appears in Figure 5.

At the beginning of the multimedia presentation, RealVideo object 302 creates a video window for rendering video, while RealTOC creates a TOC window for rendering the table of contents. Both windows are rendered within the larger RealPlayer™ window. As the time-line increases during the presentation, RealPlayer™ 301 periodically calls SAL 310 with the current time; SAL 310 passes the current time to JVM 313, which in turn notifies RealTOC 314; RealTOC 314 highlights the appropriate node on the table of contents tree rendered in the TOC window. A similar process can be constructed with the SAL keeping current time and calling the RealPlayer™ with time updates.

When the user clicks on a particular heading within the table, RealTOC 314 sends the "begin" time associated with the

heading to the "seek" function of RealPlayer™ 301 through JVM 313 and SAL 310; RealPlayer™ 301 notifies all synchronized media servers (not shown) of the new "begin" time, waits for the all the media servers to synchronize to the new time, then updates its internal current time and sends the new current time to its extension modules, *i.e.*, RealVideo 302 and SAL 310 (and RealText 103, if applicable); SAL 310 updates the current time of RealTOC 314 through JVM module 313. Importantly, both RealTOC 314 and RealVideo 302 are both synchronized to the current time of RealPlayer™ 301. Thus, after RealTOC 314 invokes the seek function requesting a new time, its current time does not skip to the new time until RealPlayer™ 301 notifies RealTOC 314 of the change in current time. If the seek function in RealPlayer™ 301 is disabled, as it should be for live video, the time line continues uninterrupted.

Seek function is only one example of controlling the flow of data. Fast forward and rewind functions also control the flow of data by changing the rate of the flow. Moreover, as discussed previously, the data can be arranged in multiple dimensions; then, its flow direction is a vector whose direction may be controlled from the extension module in a way similar to the control of data flow from the table of contents described above.

Although a specific embodiment of this invention has been shown and described, it will be understood that various

modifications may be made without departing from the spirit of this invention.